Charlie Basantes

COP 4045 001

2/26/2020

Homework #3

Problem 1)

Write the code for this problem in a file named p1_Lastname_Firstname.py, as required above.

Implement in Python a class called NVector that represents an N-dimensional vector of real numbers. The vector elements must be stored in a list object. For example NVector(3,0,1, -1) represents the 4dimensional vector [3,0,1, -1] and its elements are stored in a standard Python list [3,0,1, -1].

The vector class must have the following methods: a) constructor. If it has only one argument (besides self) assume it is a sequence of numbers (e.g. a tuple or list) and initialize the elements from the NVector accordingly, in a new list object. E.g. n4 = NVector([3,0,1, -1]) or NVector((3,0,1, -1)), the latter using a 4-tuple as parameter.

Hints: Look up functions with arbitrary (or variable) number of arguments in the lecture notes or textbook. Do not simply save the actual parameter object as the new object's state. It is very, very wrong to share the sequence object between the actual parameter and the object. That would enable side effects, and that is error-prone. In this case the call is list(param). If param is a scalar (not a sequence, like a list), the list() constructor call will raise TypeError.

b) the constructor can also take two or more arguments (besides self) – which will be the elements of the vector. E.g. NVector(3,0,1, -1) creates a vector with length 4 (i.e. 4 dimensional) and elements 3,0,1, and -1.

c) method __len__ that returns the length of the vector. E.g. NVector([3,0,1, -1]).__len_() returns 4. Note: x.__len__() is actually invoked when we call len(x).

d) the "index operator" [i], with method __getitem__(index). The argument is an index (an int) into the objects element list. E.g. if x== NVector([3,0,1, -1]), then x[1]==0. Indexing with negative numbers should work like list indexing with standard lists: E.g. if x== NVector([3,0,1, -1]), then x[-2]==1. Note that it is not required to implement list-style slicing.

e) the indexed assignment operator [], with method __setitem__(index, value), that will assign value to the element at position index. E.g. if x== NVector([3,0,1, -1]). The call x[2] = 5 will modify the vector to be the same as NVector([3,0,5, -1]. Indexing with negative numbers should work like list indexing with standard lists.

f) __str__ method that returns the string representation of the NVector object. Pick an obvious format.

g) methods __eq__ and __ne__ that take a parameter and return true if self is equal (respectively, not equal) to the parameter object, and false otherwise. For __eq__ to return true, self must be compared to another NVector object and corresponding elements with the same index should be equal.

h) method __add__ for addition with another NVector (done element-wise) or with a number (applied to all elements), and method __radd__ implementing "reflected" addition, as described in the book. E.g. NVector([3,0,1, -1]) + NVector([1,2,3,4]) results in NVector([4, 2, 4, 3])   NVector([3,0,1, -1]) + 10 results in NVector([13, 10, 11, 9])   10 + NVector([3,0,1, -1]) results in NVector([13, 10, 11, 9])

i) methods __mul__ and __rmul__, for scalar multiplication with another NVector or a number and "reflected" multiplication. If B is a number, A * B = ∑i=0 len(A)−1 Ai∗B . If B is another NVector, A * B = ∑i=0 len(A)−1 Ai∗Bi E.g.   NVector([3,0,1, -1]) * NVector([1,2,3,4]) == 3*1+0*2+1*3+(-1)*4 == 2.

j) method zeros(n) that returns a new NVector object with dimension n with all elements 0.

k) Write a main function that uses the testif() function from Homework 3 (also posted on the homework 4's page and in the Appendix) to test the code from parts a) – i). For example, if we want to test __setitem__ we can do this: v = NVector(1,2,3,4)   v[3] = 10   testif(v[3] == 10, "setitem works", "setitem failed")

Implementation requirements: Follow these requirements to get full credit. The methods should throw standard Python exceptions, as necessary. E.g. __setitem__ with an invalid index should throw IndexError. In most cases, exceptions thrown by the standard list operations are acceptable. It is your job to think about error cases and raise the proper exception if necessary. Write a docstring for each method that describes the method's contract (preconditions, postconditions, parameters, as needed).

My solution(incomplete, missing functions):

```
# -*- coding: utf-8 -*-
"""
Created on Wed Feb 26 18:41:19 2020

@author: solid
"""

class NVector:
    '''
    a) constructor. If it has only one argument (besides self) assume it is a
sequence of numbers
    (e.g. a tuple or list) and initialize the elements from the NVector
accordingly, in a new list object.

    '''
    def __init__(self, argument_1 = tuple):
        self.list = list()
        #self.vect = []

        if type(argument_1) is list:
            self.list = argument_1
            #print(self.list)
        elif type(argument_1) is tuple:
            for i in argument_1:
                self.list.append(i)
```

```
            #print(self.list)


    '''
    b) the constructor can also take two or more arguments (besides self)
    - which will be the elements of the vector.
    huh?
    '''




    '''
    c) method __len__ that returns the length of the vector.
    E.g. NVector([3,0,1, -1]).__len_() returns 4.
    Note: x.__len__() is actually invoked when we call len(x).

    '''
    def __len__(self):
        return len(self.list)

    '''
    d) the "index operator" [i], with method __getitem__(index).
    The argument is an index (an int) into the objects element list.
    E.g. if x== NVector([3,0,1, -1]),  then x[1]==0.
    Indexing with negative numbers should work like list indexing with
standard lists:
        E.g. if x== NVector([3,0,1, -1]),  then x[-2]==1.
        Note that it is not required to implement list-style slicing.

    '''
    def ___getitem__(self, index):
        temp = abs(index)
        try:
            return self.list[temp]
        except IndexError:
            print("Index is out of range.")
            return None

    '''
    e) the indexed assignment operator [], with method __setitem__(index,
value),
    that will assign value to the element at position index.
    E.g. if x== NVector([3,0,1, -1]).
    The call x[2] = 5 will modify the vector to be the same as
NVector([3,0,5, -1].
    Indexing with negative numbers should work like list indexing with
standard lists.

    '''
    def __setitem__(self, index, value):
        temp = abs(index)

        try:
```

```
            self.list[temp] = value
            return self.list
        except IndexError:
            print("Index is out of range.")
            return None

    '''
    f) __str__ method that returns the string representation of the NVector
object.
    Pick an obvious format.
    '''

    def __str__(self):
        string_vector = "["
        for i in self.list:
            string_vector += str(i) + " "
        string_vector += "]"
        return string_vector

    '''
    g) methods __eq__ and __ne__ that take a parameter and
        return true if self is equal (respectively, not equal)
        to the parameter object, and false otherwise.
        For __eq__ to return true, self must be compared to another NVector
object and
        corresponding elements with the same index should be equal.
    '''
    def __eq__ne__(self,compare):
        if len(self.list) == len(compare):
            print("Vectors share the same length:", len(self.list), "=",
len(compare))

            if (self.list == compare.list):
                print("Vectors share same values too:", self.list, "=",
compare.list)
                return True
            else:
                print("However, their values are different:", self.list, " ",
compare.list)
                return False
        else:
            print("Vectors do not share the same length:", len(self.list),
"!=", len(compare))
            return False

    '''
    h) method __add__ for addition with another NVector (done element-wise)
        or with a number (applied to all elements),
        and method __radd__ implementing “reflected” addition, as described
in the book.
        E.g.
        NVector([3,0,1, -1]) + NVector([1,2,3,4]) results in NVector([4, 2,
4, 3])
```

```
        NVector([3,0,1, -1]) + 10 results in NVector([13, 10, 11, 9])
        10 + NVector([3,0,1, -1]) results in NVector([13, 10, 11, 9])

        not fully correct doin weird stuff
    '''
    def __add__(self, number):
        temp = self.list
        temp_2 = number.list
        other = []
        if len(self.list) == len(number):
            for i in self.list:
                other = temp[i] + temp_2[i]

        else:
            return self.list + number

        return self.list

    '''
    i) methods __mul__ and __rmul__, for scalar multiplication with another
NVector or a number
        and “reflected” multiplication.
        If B is a number, A * B = ∑i=0 len(A)−1 Ai∗B .
        If B is another NVector,  A * B = ∑i=0 len(A)−1 Ai∗Bi
        E.g.
        NVector([3,0,1, -1]) * NVector([1,2,3,4]) == 3*1+0*2+1*3+(-1)*4 == 2.

    '''
    def __mul__(self):
        return None

def main():
    '''
    n4 = NVector([3,0,1, -1]) or NVector((3,0,1, -1)), the latter using a 4-
tuple as parameter

    '''

    vector_1 = NVector([3, 0, 1, -1])
    vector_2 = NVector((3, 0, 1, -1))


    #vecter.__init__()
    print('-' * 20)
    print("test length")
    print("The length of the vector_1 is:", vector_1.__len__())
    print("The length of the vector_2 is:", vector_2.__len__())


    print('-' * 20)
    print("test getitem")
    print("vector_1[0] =", vector_1.___getitem__(0))
    print("vector_1[-2] =", vector_1.___getitem__(-2))
```

```python
    print("vector_1[4] =", vector_1.___getitem__(4))

    print('-' * 20)
    print("test getitem 2")
    print("vector_2[0] =", vector_2.___getitem__(0))
    print("vector_2[-2] =", vector_2.___getitem__(-2))
    print("vector_2[4] =", vector_2.___getitem__(4))
    print('-' * 20)

    print('-' * 20)
    print("test setitem 2")
    print(vector_1.__setitem__(3, 12))
    print(vector_1.__setitem__(4, 1))
    print(vector_2.__setitem__(2, -5))
    print(vector_2.__setitem__(0, -3))
    print('-' * 20)

    print('-' * 20)
    print("test vector to string")
    print(vector_1.__str__())
    print(vector_2.__str__())
    print('-' * 20)

    print('-' * 20)
    print("test equals")
    vector_3 = NVector([3, 0, 1])
    vector_4 = NVector([-3, 0, -5, -1])
    print(vector_2.__eq__ne__(vector_1))
    print(vector_1.__eq__ne__(vector_3))
    print(vector_2.__eq__ne__(vector_4))

    print('-' * 20)
    print('-' * 20)
    print("test to add")
    vector_5 = NVector([3, 0, 1, -1])
    vector_6 = NVector([1, 2, 3, 4])
    print(vector_5.__add__(vector_6))


if __name__ == '__main__':
    main()
```

Problem 2

Write the code for this problem in a file named p2_Lastname_Firstname.py, as required above.

An online shop sells products (class Product) that are described by name (string), mass (a number in kg), quantity in stock (integer) and the price (float, USD). The class has methods named accordingly and a method __str__ that returns a user-friendly string, as seen below.

A new version of their web software must add support for discounted products. A discounted product is also a product (subclass of Product, inheriting all methods) and adds a new attribute, the discount. The design is a bit peculiar, in the sense that the DiscountedProduct object must encapsulate a Product object on which it applies the discount. The DiscountedProduct object depends on the encapsulated Product object to implement all its methods: the name is modified (as seen below) to include the discount, the price is the Product's price minus the discount, while the mass and weight are identical. The only inherited attributes actually used in a DiscountedProduct object are the reference to the encapsulated Product and the discount.

Example of using these classes:

```
# create a product object for Lavalamps, priced at $100, and with 123 of them
in stock: p = Product(name="Lavalamp", price=30, mass=0.8, stock=123)
print(p)   # prints "Lavalamp, $30, 0.8 kg, 123 in stock"
# p.price() returns 30.0
# create a discounted  product of p, with a 20% discount: disc_p =
DiscountedProduct(0.2, p)
print(disc_p.price()) # prints "24" (24 == 30 - 20% * 30)
print(disc_p)   # prints "discounted 20%: Lavalamp, $24, 0.8 kg, 123 in stock"
# now, we change the product p: p.set_price(20)
print(p.price())    # prints "20"
# the price change also affects the discounted product object that embeds p:
print(disc_p)   # prints "discounted 20%: Lavalamp, $16, 0.8 kg, 123 in stock"
# disc_p.price() returns 16 (16 == 20 - 20% * 20)
```

Implement the Product and DiscountedProduct classes, following the requirements above, and making sure the DiscountedProduct's methods rely on the results returned by the encapsulated Product object. (This design follows the Decorator design pattern.)

Write a main function that illustrates how the two classes are used. You can start from the sample code above.

My solution:

```
# -*- coding: utf-8 -*-
"""
Created on Wed Feb 26 21:49:29 2020

@author: solid
"""

class Product:
    def __init__(self, name, mass, stock, price):
        self.name = name
        self.mass = mass
```

```python
        self.stock = int(stock)
        self.price = float(price)


    def get_price(self):
        return self.price

    def __str__(self): #method to convert object of this class to string
        name_str = self.name
        price_str = self.price.__str__()
        mass_str = self.mass.__str__()
        stock_str = self.stock.__str__()
        return name_str +", $"+ price_str +", "+ mass_str +" kg, "+ stock_str
+" in stock"

    def set_price(self, price):
        self.price = price
        return self.price


class DiscountedProduct: #parametrized constructor
    def __init__(self,disc,Product):
        self.disc=disc
        self.Product=Product

    def price(self): #getter method to get discounted price
        return self.Product.price-(self.disc*self.Product.price)


    def __str__(self): #method to convert object of this class to string
        discount = (self.disc*100).__str__()
        product_name = self.Product.name
        product_price = self.price().__str__()
        product_mass = self.Product.mass.__str__()
        in_stock = self.Product.stock.__str__()

        return "discounted "+ discount +"%: "+ product_name +", $"+
product_price +", "+ product_mass +" kg, "+ in_stock +" in stock"


def main():

    # create a product object for Lavalamps, priced at $100,
    p = Product(name  = "Lavalamps", price = 30, mass = 1.0, stock = 1)
    print(p.set_price(100))
    #and with 123 of them in stock: p = Product(name="Lavalamp", price=30,
mass=0.8, stock=123)
    p = Product(name="LavaLamps", mass = 0.8, stock = 123, price = 30)
    print(p)  # prints "Lavalamp, $30, 0.8 kg, 123 in stock"

    # p.price() returns 30.0
    print(p.get_price())
```

```
    # create a discounted  product of p, with a 20% discount:
    disc_p = DiscountedProduct(0.2, p)
    print(disc_p.price()) # prints "24" (24 == 30 - 20% * 30)

    print(disc_p)  # prints "discounted 20%: Lavalamp, $24, 0.8 kg, 123 in
stock"
    # now, we change the product p:
    p.set_price(20)

    print(p.get_price())   # prints "20"

    # the price change also affects the discounted product object that embeds
p:
    print(disc_p)      # prints "discounted 20%: Lavalamp, $16, 0.8 kg, 123
in stock"

    # disc_p.price() returns 16 (16 == 20 - 20% * 20)
    print(disc_p.price())


if __name__ == '__main__':
    main()
```

Problem 3

Write the code for this problem in a file named p3_Lastname_Firstname.py, as required above.

Take the prey-predator problem described in Chapter 13 and add humans. Start from file program1314.py, attached to the assignment page.

A human is an animal that kills and eats predators, and also moves and breeds like an animal. Humans do not kill prey.

Here are the detailed rules pertaining to humans: A Human object moves like an Animal object on the island grid. A Human object does not kill prey, in contrast to predators. A Human object eats Predator objects periodically. Every Human.hunt_time clock ticks (starting with the Human object's creation time), if a Predator is in a neighboring cell (using check_grid()), the Human will move to its cell and remove the Predator from the island, in the same way Predators eat Prey objects.

 A Human object "starves" and is removed from the island if it has not killed a predator within a starving time given by a Human.starve_time class attribute, initialized in main(). A Human object breeds like an Animal object, with the breeding period given by a Human.breed_time class attribute, initialized in main().

All other rules for Prey and Predators remain in effect.

We want to study the impact of humans on the island animal populations. Add the following to the Island class: Proper initialization for Human objects. The constructor and init_animals() should take each an extra parameter count_humans and init_animals() should position Human objects at random positions. A method count_humans() that returns the number of Human objects on the grid.

The main() method should: Take extra parameters for the Human class attributes described above and should properly initialize them. Keep track of the Human population for each clock tick in the same ways it's done for Predator and Prey objects. Stop the simulation only when all three populations converge to constant values or after a maximum of 1000 time units. Display at the end with matplotlib the evolution in time of the Prey, Predator, and Human populations. Display the island grid to the terminal, at the beginning and at the end of the simulation.

More requirements: 1. Add a Human class to the existing class hierarchy so that code is properly reused. Use the problem description above to decide what class is Human's superclass. 2. Use the object-oriented design process. Integrate class Human smoothly into the existing design and code. 3. Print a Human object on the Island grid with character 'H'. 4. Apply the proper coding style and techniques taught in this class. 5. Write docstrings for functions and comment your code following the guidelines from the textbook. 6. Run the program with different combinations of parameters and find an interesting case. 7. Take a screenshot with the matplotlib chart showing the evolution of the three populations vs. time. Insert this screenshot in the Word document. This chart looks like that on slide 46 in the Chapter 13 lecture notes PDF flle. 8. Take a screenshot with the terminal showing the island grid printout (first tick and last tick) + any other statistics displayed in main(). Insert this screenshot in the Word document.

Added changes:

```python
# -*- coding: utf-8 -*-
"""
Created on Wed Feb 26 23:02:59 2020

@author: solid
"""

# Copyright 2017, 2013, 2011 Pearson Education, Inc., W.F. Punch & R.J.Enbody
"""Predator-Prey Simulation
   four classes are defined: animal, predator, prey, and island
   where island is where the simulation is taking place,
   i.e. where the predator and prey interact (live).
   A list of predators and prey are instantiated, and
   then their breeding, eating, and dying are simulted.
"""
import random
import time
import pylab

class Island (object):
```

```
    """Island
       n X n grid where zero value indicates not occupied."""
    def __init__(self, n, prey_count=0, predator_count=0, human_count=0):
#Humans added
        '''Initialize grid to all 0's, then fill with animals
        '''
        # print(n,prey_count,predator_count)
        self.grid_size = n
        self.grid = []
        for i in range(n):
            row = [0]*n    # row is a list of n zeros
            self.grid.append(row)
        self.init_animals(prey_count,predator_count, human_count)

    def init_animals(self,prey_count, predator_count,  human_count): #Humans
added
        ''' Put some initial animals on the island
        '''
        count = 0
        # while loop continues until prey_count unoccupied positions are
found
        while count < prey_count:
            x = random.randint(0,self.grid_size-1)
            y = random.randint(0,self.grid_size-1)
            if not self.animal(x,y):
                new_prey=Prey(island=self,x=x,y=y)
                count += 1
                self.register(new_prey)
        count = 0
        # same while loop but for predator_count
        while count < predator_count:
            x = random.randint(0,self.grid_size-1)
            y = random.randint(0,self.grid_size-1)
            if not self.animal(x,y):
                new_predator=Predator(island=self,x=x,y=y)
                count += 1
                self.register(new_predator)

        ##added Human counter as instructed
        while count < human_count:
            x = random.randint(0,self.grid_size-1)
            y = random.randint(0,self.grid_size-1)
            if not self.animal(x,y):
                new_human=Human(island=self,x=x,y=y)
                count += 1
                self.register(new_human)

    def clear_all_moved_flags(self):
        ''' Animals have a moved flag to indicated they moved this turn.
        Clear that so we can do the next turn
        '''
        for x in range(self.grid_size):
            for y in range(self.grid_size):
```

```python
            if self.grid[x][y]:
                self.grid[x][y].clear_moved_flag()

    def size(self):
        '''Return size of the island: one dimension.
        '''
        return self.grid_size

    def register(self,animal):
        '''Register animal with island, i.e. put it at the
        animal's coordinates
        '''
        x = animal.x
        y = animal.y
        self.grid[x][y] = animal

    def remove(self,animal):
        '''Remove animal from island.'''
        x = animal.x
        y = animal.y
        self.grid[x][y] = 0

    def animal(self,x,y):
        '''Return animal at location (x,y)'''
        if 0 <= x < self.grid_size and 0 <= y < self.grid_size:
            return self.grid[x][y]
        else:
            return -1  # outside island boundary

    def __str__(self):
        '''String representation for printing.
           (0,0) will be in the lower left corner.
        '''
        s = ""
        for j in range(self.grid_size-1,-1,-1):  # print row size-1 first
            for i in range(self.grid_size):      # each row starts at 0
                if not self.grid[i][j]:
                    # print a '.' for an empty space
                    s+= "{:<2s}".format('.' + "  ")
                else:
                    s+= "{:<2s}".format((str(self.grid[i][j])) + "  ")
            s+="\n"
        return s

    def count_prey(self):
        ''' count all the prey on the island'''
        count = 0
        for x in range(self.grid_size):
            for y in range(self.grid_size):
                animal = self.animal(x,y)
                if animal:
                    if isinstance(animal,Prey):
                        count+=1
```

```
        return count

    def count_predators(self):
        ''' count all the predators on the island'''
        count = 0
        for x in range(self.grid_size):
            for y in range(self.grid_size):
                animal = self.animal(x,y)
                if animal:
                    if isinstance(animal,Predator):
                        count+=1
        return count

    #added Human counter as instructed
    def count_humans(self):
        ''' count all the human on the island'''
        count = 0
        for x in range(self.grid_size):
            for y in range(self.grid_size):
                animal = self.animal(x,y)
                if animal:
                    if isinstance(animal,Human):
                        count+=1
        return count

class Animal(object):
    def __init__(self, island, x=0, y=0, s="A"):
        '''Initialize the animal's and their positions
        '''
        self.island = island
        self.name = s
        self.x = x
        self.y = y
        self.moved=False

    def position(self):
        '''Return coordinates of current position.
        '''
        return self.x, self.y

    def __str__(self):
        return self.name

    def check_grid(self,type_looking_for=int):
        ''' Look in the 8 directions from the animal's location
        and return the first location that presently has an object
        of the specified type. Return 0 if no such location exists
        '''
        # neighbor offsets
        offset = [(-1,1),(0,1),(1,1),(-1,0),(1,0),(-1,-1),(0,-1),(1,-1)]
        result = 0
        for i in range(len(offset)):
            x = self.x + offset[i][0]  # neighboring coordinates
```

```
            y = self.y + offset[i][1]
            if not 0 <= x < self.island.size() or \
               not 0 <= y < self.island.size():
                continue
            if type(self.island.animal(x,y))==type_looking_for:
                result=(x,y)
                break
        return result

    def move(self):
        '''Move to an open, neighboring position '''
        if not self.moved:
            location = self.check_grid(int)
            if location:
                # print('Move, {}, from {},{} to {},{}'.format( \
                #        type(self),self.x,self.y,location[0],location[1]))
                self.island.remove(self)   # remove from current spot
                self.x = location[0]       # new coordinates
                self.y = location[1]
                self.island.register(self) # register new coordinates
                self.moved=True
    def breed(self):
        ''' Breed a new Animal.If there is room in one of the 8 locations
        place the new Prey there. Otherwise you have to wait.
        '''
        if self.breed_clock <= 0:
            location = self.check_grid(int)
            if location:
                self.breed_clock = self.breed_time
                # print('Breeding Prey {},{}'.format(self.x,self.y))
                the_class = self.__class__
                new_animal =
the_class(self.island,x=location[0],y=location[1])
                self.island.register(new_animal)

    def clear_moved_flag(self):
        self.moved=False

class Prey(Animal):
    def __init__(self, island, x=0,y=0,s="O"):
        Animal.__init__(self,island,x,y,s)
        self.breed_clock = self.breed_time
        # print('Init Prey {},{}, breed:{}'.format(self.x,
self.y,self.breed_clock))

    def clock_tick(self):
        '''Prey only updates its local breed clock
        '''
        self.breed_clock -= 1
        # print('Tick Prey {},{},
breed:{}'.format(self.x,self.y,self.breed_clock))

class Predator(Animal):
```

```
    def __init__(self, island, x=0,y=0,s="X"):
        Animal.__init__(self,island,x,y,s)
        self.starve_clock = self.starve_time
        self.breed_clock = self.breed_time
        # print('Init Predator {},{}, starve:{}, breed:{}'.format( \
        #       self.x,self.y,self.starve_clock,self.breed_clock))

    def clock_tick(self):
        ''' Predator updates both breeding and starving
        '''
        self.breed_clock -= 1
        self.starve_clock -= 1
        # print('Tick, Predator at {},{} starve:{}, breed:{}'.format( \
        #       self.x,self.y,self.starve_clock,self.breed_clock))
        if self.starve_clock <= 0:
            # print('Death, Predator at {},{}'.format(self.x,self.y))
            self.island.remove(self)

    def eat(self):
        ''' Predator looks for one of the 8 locations with Prey. If found
        moves to that location, updates the starve clock, removes the Prey
        '''
        if not self.moved:
            location = self.check_grid(Prey)
            if location:
                # print('Eating: pred at {},{}, prey at {},{}'.format( \
                #       self.x,self.y,location[0],location[1]))

self.island.remove(self.island.animal(location[0],location[1]))
                self.island.remove(self)
                self.x=location[0]
                self.y=location[1]
                self.island.register(self)
                self.starve_clock=self.starve_time
                self.moved=True

#Added Human class as required
class Human(Animal):
    def __init__(self, island, x=0,y=0,s="H"):
        Animal.__init__(self,island,x,y,s)
        self.starve_clock = self.starve_time
        self.breed_clock = self.breed_time
        # print('Init Predator {},{}, starve:{}, breed:{}'.format( \
        #       self.x,self.y,self.starve_clock,self.breed_clock))

    def clock_tick(self):
        ''' Predator updates both breeding and starving
        '''
        self.breed_clock -= 1
        self.starve_clock -= 1
        # print('Tick, Predator at {},{} starve:{}, breed:{}'.format( \
        #       self.x,self.y,self.starve_clock,self.breed_clock))
        if self.starve_clock <= 0:
```

```
            # print('Death, Predator at {},{}'.format(self.x,self.y))
            self.island.remove(self)

    def eat(self):
        ''' Human looks for one of the 8 locations with Prey. If found
        moves to that location, updates the starve clock, removes the Prey
        '''
        if not self.moved:
            location = self.check_grid(Predator)
            if location:
                # print('Eating: pred at {},{}, prey at {},{}'.format( \
                #       self.x,self.y,location[0],location[1]))

self.island.remove(self.island.animal(location[0],location[1]))
                self.island.remove(self)
                self.x=location[0]
                self.y=location[1]
                self.island.register(self)
                self.starve_clock=self.starve_time
                self.moved=True

###########################################
def main(human_breed_time = 6, human_starve_time = 3, intial_humans = 10,
predator_breed_time=6, predator_starve_time=3, initial_predators=10,
prey_breed_time=3, initial_prey=50, \
         size=10, ticks=300):
    ''' main simulation. Sets defaults, runs event loop, plots at the end
    '''
    # initialization values
    Predator.breed_time = predator_breed_time
    Predator.starve_time = predator_starve_time
    Prey.breed_time = prey_breed_time

    # for graphing
    predator_list=[]
    prey_list=[]
    human_list=[] #added humans

    # make an island
    isle = Island(size,initial_prey, initial_predators, intial_humans)
    print(isle)

    # event loop.
    # For all the ticks, for every x,y location.
    # If there is an animal there, try eat, move, breed and clock_tick
    for i in range(ticks):
        # important to clear all the moved flags!
        isle.clear_all_moved_flags()
        for x in range(size):
            for y in range(size):
                animal = isle.animal(x,y)
                if animal:
                    if isinstance(animal,Predator):
```

```
                        animal.eat()
                        if isinstance(animal,Human):
                            animal.eat()
                    animal.move()
                    animal.breed()
                    animal.clock_tick()

        # record info for display, plotting
        prey_count = isle.count_prey()
        predator_count = isle.count_predators()
        human_count = isle.count_humans()
        if prey_count == 0:
            print('Lost the Prey population. Quiting.')
            break
        if predator_count == 0:
            print('Lost the Predator population. Quitting.')
            break
        if human_count == 0:
            print('Lost the Human population. Quitting.')
            break
        prey_list.append(prey_count)
        predator_list.append(predator_count)
        human_list.append(human_count)
        # print out every 10th cycle, see what's going on
        if not i%10:
            print(prey_count, predator_count, human_count)
        # print the island, hold at the end of each cycle to get a look
#        print('*'*20)
#        print(isle)
#        ans = input("Return to continue")
    pylab.plot(range(0,ticks), predator_list, label="Predators")
    pylab.plot(range(0,ticks), prey_list, label="Prey")
    pylab.plot(range(0,ticks), human_list, label="Human")
    pylab.legend(loc="best", shadow=True)
    pylab.show()

if __name__ == '__main__':
    main()
```

Screenshot I have:

Something failed, unsure what did, still working on.

```
In [95]: runfile('D:/COP4045 Python/Hw3/p3_Basantes_Charlie.py', wdir='D:/COP4045 Python/Hw3')
.  .  O  O  O  .  .  .  O  .
.  .  X  O  X  .  .  O  .  .
X  O  X  .  O  O  .  X  .  .
O  O  O  O  O  X  O  O  O  .
O  X  O  .  .  O  O  O  O  O
O  .  .  O  O  .  .  .  O  X
O  .  O  .  O  X  .  .  O  O
O  .  O  .  O  O  .  O  .  O
O  O  .  .  .  O  O  O  .  .
.  O  O  O  .  O  .  O  X  O

Lost the Human population. Quitting.
Traceback (most recent call last):
```

```
ValueError: x and y must have same first dimension, but have shapes (300,) and (0,)
```